Модел. и анализ информ. систем. Т. **18**, № **1** (2011) 116–129

УДК 681.3

Стратегия выполнения операций копирования и удаления в дереве объектов

Майоров А. В.

ООО "Байт-Форс"

e-mail: xor@byte-force.com

получена 18 сентября 2010

Ключевые слова: дерево объектов, граф объектов, копирование, удаление, реляционная база данных

Описываются общие правила организации объектов в иерархические структуры. Приводятся различные типы связей, объясняется их роль в организации объектов. Рассматриваются принципы выполнения операций копирования и удаления объектов, не нарушающих логику связей между объектами.

1. Введение

Организация объектов предметной области в древовидную структуру удобна при проектировании приложений, потому что многие моделируемые сущности реального мира сами имеют иерархическую структуру. При этом у объекта часто бывает больше одного родителя, поэтому удобно использовать не дерево (в строгой его трактовке), а направленный ациклический граф.

В такой структуре очень просто реализуется операция добавления нового объекта, и почти также прост перенос объекта от одного родителя к другому. Операции же копирования и удаления объектов уже не так очевидны.

Мы детально рассмотрим, как должна проводиться операция копирования объекта, какие из его дочерних объектов затрагиваются, и как должны копироваться связи этих объектов с другими. Для операции удаления мы также исследуем ситуацию, когда объект не может быть удален из-за входящих ссылок, но пользователь настаивает на удалении.

При этом мы будем ориентироваться на наиболее привычное для пользователя поведение системы, когда, например, копирование контейнера копирует также и его содержимое.

2. Организация иерархического хранилища данных

Иерархическая модель хранения данных — довольно старая концепция. Иерархические СУБД были популярны в конце 60-х годов прошлого века. Среди них наиболее известна Information Management System, разработанная компанией IBM. Система до сих пор развивается и успешно продается.

Классические иерархические СУБД организуют информацию в набор родительских и дочерних сегментов. Эта структура подразумевает, что записи могут иметь повторяющиеся фрагменты. Особенно в дочерних сегментах. Все экземпляры записей с определенным набором полей образуют тип записи. Тип записи — эквивалент таблицы в реляционной базе данных, в то время как запись — эквивалент строки таблицы (то есть записи) [1].

В данной статье мы рассматриваем не способы хранения деревьев, а стратегию выполнения элементарных операций над объектами в дереве. Поэтому особенности хранения нас сейчас не интересуют. Предположим, например, что иерархия объектов содержится в обычной реляционной СУБД.

При этом важно помнить, что мы работаем с иерархией объектов, а не записей. Объект — несколько более сложная сущность, чем запись в таблице или иерархической БД. Объект является логическим объединением нескольких записей реляционной БД, с которыми система работает как с единым блоком информации [2].

3. Объекты в системе

Все данные в системе делятся на две основные категории: объекты, принадлежащие иерархии, и все остальное. Про первую категорию мы и будем говорить в этой статье. Для простоты будем называть объекты из иерархии просто «объектами». В категорию «остальное» входят метаданные, необходимые для функционирования системы, и разнообразные справочники.

4. Типы объектов

В системе может быть любое количество различных типов объектов. При этом мы считаем, что все они произведены от единого базового типа и разделяют его свойства.

Перечислим несколько распространенных типов объектов:

- Корневой объект — специальный тип для корневого объекта.
- Персона. Объект этого типа представляет пользователя системы или просто человека, о котором в системе есть данные.
- Папка. Объекты этого типа служат для структурирования информации в базе. Аналог каталогов на диске.
- Статья. Объекты этого типа имеют заголовок, текст и т.п. Все текстовые поля могут быть переведены на несколько языков.

В целом, система могла бы успешно функционировать и в том случае, если бы в ней был зарегистрирован только один универсальный тип объектов. Возможно, для каких-то приложений это имеет смысл.

5. Граф объектов

В предлагаемой модели, все объекты системы связаны между собой. Связи типа “parent-child” (родитель-потомок) организуют объекты системы в направленное дерево. Это значит, что у каждого объекта, кроме корневого, имеется один родитель и любое количество потомков. Мы будем считать, что само дерево «растет» вниз, его корень вверху, а связи в нем ориентированы снизу вверх — от потомков к родителям.

Помимо связей “parent-child”, между объектами могут быть созданы и дополнительные связи других типов: “link” и “aggregate”. Этими дополнительными типами связи объект может быть связан с любым количеством родительских объектов.

Должны выполняться два условия. Во-первых, между любыми двумя объектами может быть проложена только одна связь. Вне зависимости от типа связи. Во-вторых, связи не должны образовывать циклов. Таким образом, все объекты и отношения образуют в направленный ациклический граф.

Нетрудно заметить, что подобная организация очень похожа на файловую систему. Директории и файлы образуют дерево, дополнительно там есть symbolic и hard links. В принципе, обсуждаемую модель можно рассматривать как использование парадигмы папок и файлов для проектирования приложений.

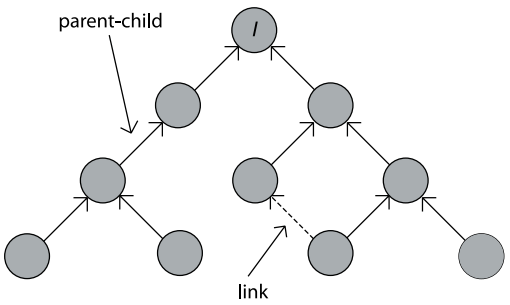

Рис. 1.

Отметим, что, если подобная иерархическая структура хранится в реляционной базе данных, то для эффективной выборки записей из ветвей дерева можно использовать метод построения иерархического индекса [2].

6. Контейнеры

Объект является контейнером, если у него есть или могут быть дочерние объекты, связанные отношением “parent-child”. В нашей модели любой объект является контейнером. Мы будем говорить, что объект A лежит внутри объекта B, если B является родителем A с отношением типа “parent-child”.

Очевидно, что дерево объектов можно также представить и в виде системы вложенных контейнеров. Корневой объект — это контейнер, вмещающий контейнеры второго уровня, которые вмещают контейнеры третьего уровня и так далее. Корневой контейнер опосредованно содержит в себе все объекты системы.

7. Поведение различных типов связей

Введем определения. **Родительскими связями** или отношениями будем называть те связи, которые связывают объект с его родителями. По любому типу связи. **Дочерние отношения** (связи) связывают объект с его дочерними объектами. Тип связи здесь роли не играет.

7.1. Parent-child

Связи типа “parent-child” организуют все объекты системы в дерево. Из любого объекта системы должна быть возможность дойти до единого корневого объекта, передвигаясь только по связям “parent-child”.

При удалении объекта, у которого есть дочерние объекты, присоединенные связью “parent-child”, мы должны удалить и все эти дочерние объекты. Удаление одного этого объекта невозможно, так как привело бы к появлению объектов, не связанных с корневым объектом системы. Важно помнить, что дочерние объекты являются самостоятельными информационными сущностями, и их автоматическое удаление может не входить в планы пользователя.

На рис. 2 показано, как удаление объекта ведет к удалению всей ветви дерева.

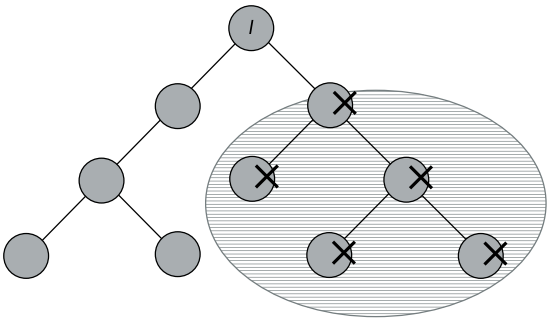

Рис. 2.

Теперь рассмотрим вопрос копирования объектов. Копирование объектов в базе данных — это не такая очевидная задача, как, например, копирование файлов. Файл на диске представляет собой совершенно независимую единицу, он никак не связан с другими файлами. Есть только связь с каталогом, в котором этот файл находится. Объект в базе данных, напротив, может иметь множество связей с другими объектами. Причем часто это не обсуждаемые нами отношения между родителем и потомком, а прямые ассоциативные связи между объектами.

Как в данном случае должно происходить копирование объекта? Копировать ли при этом и связанные объекты или сделать так, чтобы копия ссылалась на те же экземпляры, что и оригинал? Очевидно, что в общем случае эта задача не решается: все зависит от конкретной ситуации, от конкретной структуры данных.

Мы можем вывести только следующее правило: если копируется объект, у которого есть дочерние объекты, присоединенные связью “parent-child”, мы должны также скопировать эти дочерние объекты и присоединить их к копии родительского объекта (рис. 3).

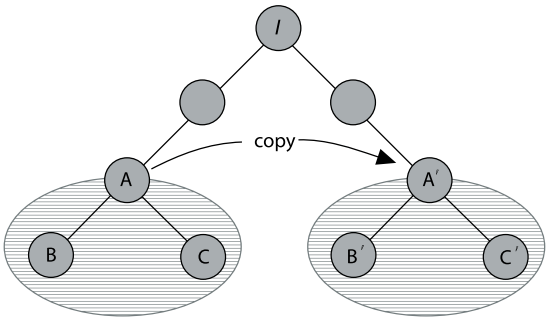

Рис. 3.

7.2. Link

Связь типа “link” позволяет проложить дополнительные пути в графе объектов. При удалении объекта, у которого есть дочерний объект, подключенный связью типа “link”, мы должны удалить только саму связь, не трогая дочерний объект.

Причина очевидна. По соглашению, у этого объекта также есть и связь типа “parent-child” с его родительским контейнером, поэтому удаление связи “link” не приведет к выпадению объекта из дерева (рис. 4).

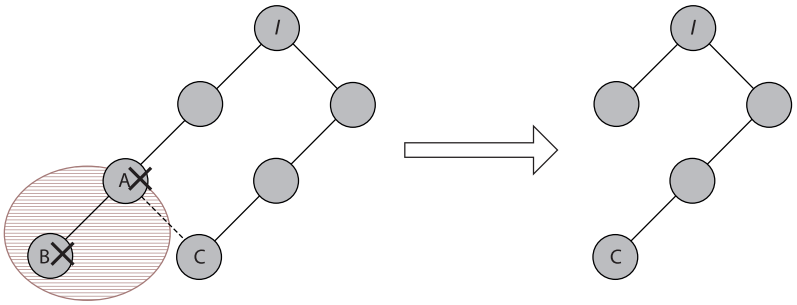

Рис. 4.

При копировании объекта, у которого есть дочерний объект, присоединенный связью типа “link”, мы должны скопировать только объект связи, не копируя сам дочерний объект. В результате копия будет связана с тем же самым дочерним объектом, что и оригинал (рис. 5).

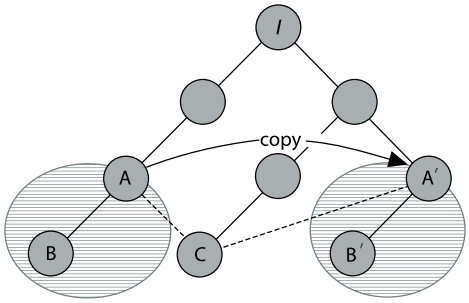

Рис. 5.

7.3. Aggregate

Связь типа “aggregate” показывает, что дочерний объект является составной частью родительского объекта.

Для иллюстрации такого типа связи представим систему управления библиотеками, в которой есть единый реестр книг и несколько библиотек. Каждая библиотека управляет своим фондом, создавая формуляры для книг. Очевидно, что формуляр без книги не имеет большого смысла. Поэтому если книга и формуляр представлены отдельными объектами, то вместе они составляют единый агрегат (и связаны соответствующим отношением).

При удалении объекта, у которого есть агрегированные дочерние объекты, мы можем без запроса удалять эти дочерние объекты. При копировании — копировать дочерние объекты вместе с родительским. На рис. 6 показано копирование книги K с формуляром Φ, принадлежащим библиотеке $Б$. Копии книги и формуляра — K' и Φ' соответственно.

Заметим, что в приведенном примере у формуляров должна быть также и связь типа “parent-child” с каким-то родительским объектом. Этого требуют общие правила нашей модели. Наиболее логично считать, что родительским объектом объекта «формуляр» является объект «библиотека». Тогда при копировании библиотеки мы скопируем все ее формуляры, а при копировании книги — все ее формуляры во всех библиотеках.

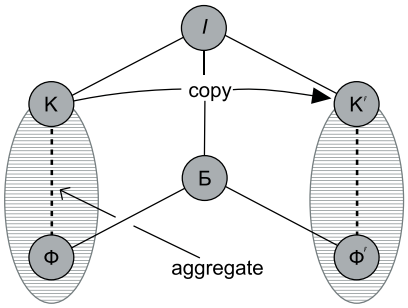

Рис. 6.

7.4. Ассоциативная связь между объектами

Мы уже упоминали, что, кроме рассматриваемых нами отношений между объектами, в базе данных могут существовать и обычные ассоциативные связи. Для реляционной базы — связи через внешние ключи. Такие связи совершенно легитимны, но находятся вне обсуждаемой сейчас модели. Описанные правила копирования и удаления не могут быть автоматически к ним применены.

С другой стороны, большинство таких связей можно отнести к одной из перечисленных категорий: “parent-child”, “link” или “aggregate”. Так что, при определении поведения системы во время удаления или копирования объектов, связанных ассоциативной связью, вполне можно руководствоваться теми же соображениями.

8. Последовательность копирования объектов

При копировании объекта создается новый экземпляр того же типа, что и оригинальный объект. Все свойства нового объекта, за исключением идентификатора, устанавливаются в те же значения, что и аналогичные свойства оригинального объекта. Если значение свойства не скалярное, то производится «глубокое» копирование этого значения.

8.1. Копирование связанных объектов

После копирования самого объекта нужно разобраться, что делать с его дочерними объектами. Предлагается такая стратегия:

- Дочерние объекты, связанные с родителем связями “parent-child” и “aggregate”, также копируются.
- Дочерние объекты, связанные связью “link”, сами не копируются. Вместо этого копия родительского объекта должна будет иметь связь типа “link” с тем же самым дочерним объектом. Другими словами, у дочернего объекта появится еще одна связь «вверх» — к копии родительского объекта.

Копирование каждого дочернего объекта, в свою очередь, заставляет нас копировать его дочерние объекты и так далее.

Введем определение.

Объект, с которого копирование было начато, будем называть ***первичным*** *копируемым объектом. Его потомков, копируемых не по прямому запросу пользователя, а потому что того требуют описанные нами правила, назовем* ***вторичными*** *копируемыми объектами. Множество всех копируемых объектов состоит из первичного объекта, объединенного с множеством вторичных объектов.*

8.2. Копирование родительских связей первичного объекта

Что должно происходить с родительскими связями первичного копируемого объекта? Ведь очевидно, что копируемый объект не только может иметь дочерние объекты, но и сам связан со своими родителями.

8.3. Родительская связь типа “parent-child”

При копировании свежая копия объекта должна быть присоединена к дереву объектов. Так что у нее точно будет один родительский объект, связанный с ней отношением “parent-child”. При этом родительский объект может быть любым – и тем, внутри которого находится оригинал, и совершенно другим – зависит только от пользователя. Тут есть прямая аналогия с копированием файлов: только пользователь определяет, в какой каталог будет положена копия файла; от самого файла или от местоположения оригинала в файловой системе это никак не зависит.

Таким образом, родительское отношение “parent-child” первичного объекта копировать не надо (рис. 7).

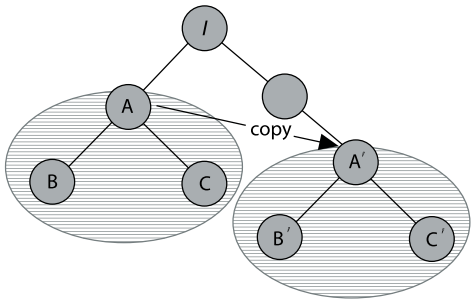

Рис. 7.

8.4. Родительская связь типа “link”

Если у первичного объекта есть родительский объект, связанный отношением “link”, то это не означает, что первичная копия также должна быть присоединена к этому родителю. Напротив, такое поведение было бы странным, так как пользователь ожидает, что в процессе копирования новый дочерний объект появится только у того объекта, в который кладется копия (рис. 8).

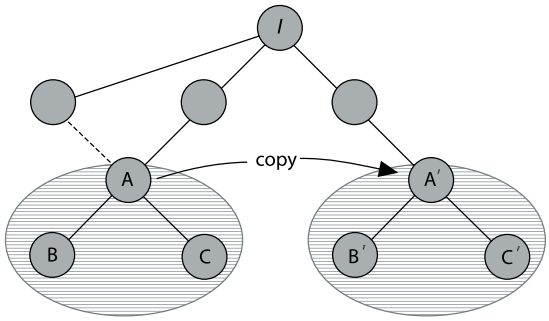

Рис. 8.

8.5. Родительская связь типа “aggregate”

Отношение типа “aggregate” показывает, что дочерний объект является составной (подчиненной) частью родительского. Что должно происходить, если первичный копируемый объект как раз является такой подчиненной частью? Логично, чтобы копия первичного объекта тоже была подчиненной частью. Это значит, что нам придется скопировать и оригинальное родительское отношение типа “aggregate”, так что копия будет являться составной частью того же самого объекта.

Вспомним пример с формулярами книг. Формуляр принадлежит библиотеке, но является составной частью книги. С библиотекой связь “parent-child”, с книгой — “aggregate”. Если мы копируем формуляр в другую библиотеку, то получаем формуляр для этой же книги в другой библиотеке. Если копия формуляра будет положена в эту же библиотеку, то мы просто получим еще один формуляр для этого же наименования книги в этой библиотеке.

Таким образом, мы видим, что при копировании первичного объекта родительские отношения типа “aggregate” нужно копировать.

8.6. Копирование родительских связей во множестве вторичных объектов

Покажем, что правила копирования родительских связей первичного копируемого объекта отличаются от правил копирования связей для вторичных объектов в этой же операции копирования.

Само появление множества вторичных объектов связано с тем, что при копировании мы обязаны воссоздать структуру объектов, зависящих от первичного объекта. Из-за этого мы не можем рассматривать копирование любого объекта из множества копируемых объектов как независимую операцию, ведь тогда могут потеряться связи между объектами внутри множества.

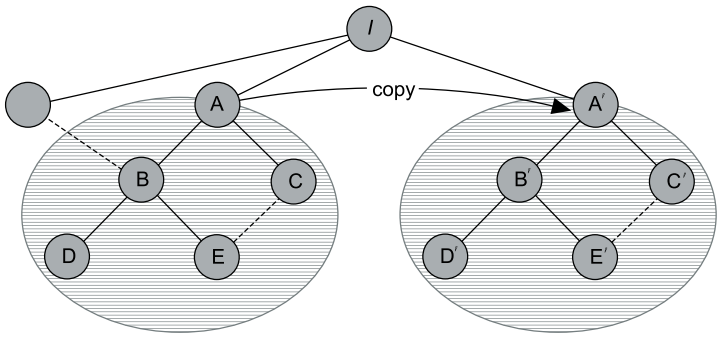

Рис. 9.

Приведем пример. Если мы копируем ветвь дерева объектов, внутри которой два объекта связаны отношением типа “link”, то мы должны повторить это отношение. Так что если оригинальные объекты C и E были связаны, то и их копии C' и E' тоже должны быть связаны между собой. В то же время, если у объекта была связь типа

“link” с объектом вне множества копируемых объектов, то эта связь не копируется (рис. 9). То есть для внешних родительских связей действуют те же правила, что и для родительских связей первичного объекта.

8.7. Отношения “parent-child” внутри множества

В начале копирования мы задаем только целевой контейнер для копии первичного объекта, все остальные родительские отношения “parent-child” должны быть воссозданы между копиями оригинальных объектов, связанных такими отношениями. Это самый очевидный случай копирования отношений.

Если объекты A и B входят во множество копируемых объектов и между ними есть отношение типа “parent-child”, то и между их копиями A' и B' должно быть установлено отношение типа “parent-child” аналогичного направления.

8.8. Родительское отношение “parent-child”, ведущее вне множества

Объект может попасть во множество вторичных объектов из-за того, что у него есть родительская связь типа “aggregate” с объектом из множества копируемых объектов. При этом его непосредственный родительский объект может и не входить во множество. В этом случае копия объекта должна быть связана с тем же самым родительским объектом, что и оригинал.

В качестве примера приведем все ту же библиотеку с формулярами. Если мы копируем книгу, у которой есть формуляры, то мы должны скопировать и формуляры. Новые формуляры будут относиться к той же библиотеке, что и оригинальные.

Итак, если объект A является дочерним к B с отношением “parent-child”, объект A принадлежит множеству копируемых объектов, а объект B — не принадлежит, то копия A' также должна быть дочерней к объекту B с тем же типом отношения.

8.9. Отношение “link” внутри множества

Этот вариант уже обсуждался на примере выше. Внутри множества отношения типа “link” должны копироваться. Другими словами, если между объектами A и B есть связь типа “link” и эти объекты принадлежат множеству копируемых объектов, то между их копиями A' и B' также должна быть создана связь типа “link” с аналогичным направлением.

8.10. Отношение типа “link” с объектом вне множества

Если объект A является дочерним к объекту B с отношением “link”, объект A принадлежит к множеству вторичных копируемых объектов, а B — не принадлежит, то связь между копией A' и B создавать не нужно.

8.11. Отношение “aggregate” внутри множества

Эта ситуация уже прорабатывалась на примере с библиотеками. Если копируется книга, к которой привязан формуляр, то к копии книги должна быть привязана копия формуляра.

На рис. 10 показано копирование библиотеки $Б$ в $Б'$. Есть книга $К$ и ее формуляр $Ф$ в первой библиотеке. Результирующая копия формуляра $Ф'$ находится во второй библиотеке и также привязана отношением aggregate к книге. Это очень похоже на рис. 6, только там мы копировали книгу, а не библиотеку.

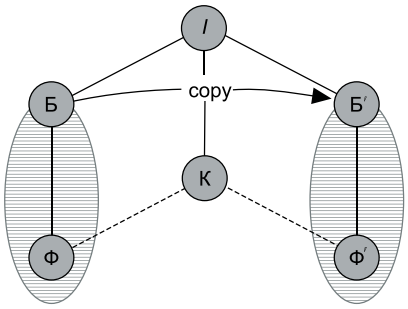

Рис. 10.

Таким образом, если между объектами A и B есть связь типа “aggregate” и эти объекты принадлежат множеству копируемых объектов, то между их копиями A' и B' также должна быть создана связь типа “aggregate” с аналогичным направлением.

8.12. Отношение “aggregate”, ведущее вне множества

Снова вспомним пример с формулярами. При копировании библиотеки мы должны скопировать все ее формуляры. При этом копии формуляров будут связаны с теми же книгами, с которыми были связаны оригиналы.

Если объект A является дочерним к B с отношением “aggregate”, объект A принадлежит множеству копируемых объектов, а объект B — не принадлежит, то копия A' также должна быть дочерней к объекту B с тем же типом отношения.

9. Порядок удаления объектов

При удалении объекта мы сначала должны проанализировать его зависимости с другими объектами. Если у объекта есть дочерние объекты, то мы не имеем права удалить их родителя, оставив их висеть «в воздухе», не прикрепленными к дереву объектов. Так что мы должны удалить либо все затронутые объекты, либо ничего.

9.1. Удаление подчиненных объектов

Как и для операции копирования, введем понятия **первичного удаляемого** объекта и **вторичных удаляемых** объектов. Первичный объект — это тот, удаление которого было запрошено пользователем. Вторичные объекты — это те, которые

должны быть удалены, потому что они не могут существовать без первичного объекта. Отбор объектов во множество происходит ровно по тем же правилам, что и при операции копирования.

Если объект попал во множество удаляемых объектов, то наличие у него родительских связей с объектами вне множества не играет никакой роли. Если, например, мы удаляем книгу, то соответственно должны будем удалить и ее формуляры. При этом каждый формуляр связан отношением “parent-child” со своей библиотекой. Существование этой связи не ограничивает пользователя в удалении книги или ее формуляра. Следовательно, при удалении объекта мы просто удаляем все его родительские связи.

После того как множество вторичных удаляемых объектов сформировано, можно приступать к их удалению, и вот здесь могут возникнуть сложности. Дело в том, что помимо отношений «родитель-потомок», объекты могут быть связаны и ассоциативными связями. Причем такие связи могут быть не только прямые, когда удаляемый объект ссылается куда-то, но и обратные, когда ссылаются именно на этот удаляемый объект.

В этом случае система контроля ссылочной целостности (reference integrity) базы данных не даст удалить те записи, на которые есть входящие ссылки. Это будет означать, что и весь объект, частью которого являются указанные записи, не может быть удален.

Далее, если какой-то объект из множества вторичных объектов не может быть удален, то нельзя удалять и его родительский объект, и родителя его родителя, и так далее, вплоть до первичного удаляемого объекта, который, как оказывается, тоже нельзя удалять.

9.2. Удаление через пометку

Очевидно, что ситуация, когда удаление объектов невозможно, так как оно приведет к нарушению ссылочной целостности, не является чем-то необычным. И при этом система все равно должна пытаться удовлетворить запрос пользователя на удаление. В таких случаях, вместо того чтобы удалять объекты из базы данных, мы просто помечаем их как удаленные.

Для пометки объекта удаленным вводим специальное свойство объекта — флаг «объект удален». Таким образом, объекты из множества удаляемых объектов делятся на три категории:

1. Те, которые можно удалить обычным способом.
2. Удаляемые через пометку, потому что на них есть внешние ссылки.
3. Удаляемые через пометку, потому что у них есть дочерние элементы, связанные отношением “parent-child” или “aggregate”, которые в свою очередь удаляются через пометку.

Объекты первой группы особого интереса не представляют — удаляется сам объект, все его родительские и дочерние связи. А вот при удалении через пометку связи нужно обрабатывать более аккуратно. Затронуты будут только те связи,

которые ведут вовне множества помечаемых объектов. Внутренние связи остаются без изменений. Постольку, поскольку для внешних объектов не должно быть разницы между обычным удалением и удалением через пометку, все внешние связи удаляются.

Получается изолированная ветвь дерева, где все объекты помечены удаленными. Корневой объект ветви — первичный удаляемый объект. Для удобства ветвь можно положить в специальный контейнер, играющий в системе роль «мусорной корзины».

Последний аспект, на который нужно обратить внимание, касается восстановления объектов из корзины. Когда пользователь восстанавливает объект, он ожидает, что объект вернется в то же место иерархии, откуда был удален. Более того, если у объекта были какие-то дополнительные связи с другими объектами, то эти связи тоже должны быть по возможности восстановлены. Мы говорим «по возможности», потому что объекты, с которыми восстанавливаемый объект был связан ранее, могут и сами уже отсутствовать. Эти связи, конечно, восстановить не удастся.

9.3. Заключение

Анализируя стратегии выполнения операций копирования и удаления, мы видим, что в предложенной модели связи типа “parent-child” и “aggregate” ведут себя очень похоже. Единственная разница между ними обусловливается требованием к формированию отношениями типа “parent-child” строгого дерева объектов. Если мы снимем это требование, то можно будет оставить только два типа связей: “parent-child” и “link”. Другими словами: «жесткая» и «мягкая» связи.

Список литературы

1. Wells April J. Grid database design. s.l. : Auerbach Pulications, Taylor & Francis Group, 2005.

2. Bancilhon F., Delobel C., Kanellakis P. Building an Object-Oriented Database. s.l.: Morgan Kaufmann, 1992.

3. Майоров А. Построение индекса по иерархии записей в реляционной базе данных // Материалы конференции SEF-2009. Минск, 2009.

A strategy for the execution of copy and delete operations in the tree of objects

Mayorov A. V.

Keywords: objects tree, objects graph, copy operation, delete operation, relation database

The paper discusses some general rules for organizing database objects into hierarchical structures. It describes different kinds of relations between objects, and the role that relations play in the organization of such structures. It also defines principles of executing object copy and delete operations that do not break the logic of relations between objects.

Сведения об авторе:
Андрей Вячеславович Майоров,
ООО "Байт-Форс", директор